ORACLE DATABASE INTERFACE PROGRAMMING
for the
Student Polarimeter Aerosol and Cloud Experiment

Torren Hope and Nigel Phillip

Mentors: Prof. Fred Moshary
Department of Physical, Environmental and Computer Sciences, Medgar Evers College, City University of New York

An interface application allows the user to access a product through easy to use commands. In the case of a database interface application the user can access the database without the need to learn structured query language (SQL), or how the database operates. This application, which is in its concept phase, is different in that we want to access the database using more complex operations than can be accomplished using SQL. The application inserts binary data into the database. The database is Oracle 8*i* installed on a Solaris Ultra 10 system. While, the data consists of Level-0 binary data on cdrom from the NASA Student Polarimeter Aerosol and Cloud Experiment. This paper will describe the binary data in more detail, the description and development of the application, and the stages of getting the data into the database.

The binary data contains a description of the data, time, date, offset information, information about the instruments used, information about the scan, and the actual scan data. This organization of the data makes creating a structure for it within an application somewhat easy. At the time this application was being developed, the binary data was stored on cdrom. Eventually the data will be transported over the network to the computer systems. In the mean time, the data files were copied off the cdroms onto the computer system were the application would reside. This allowed for faster and easier access to the data

This application was developed because there is no way to do data manipulation on binary large objects (BLOBs) in Oracle's *8i* database using SQL. Since you cannot use SQL to manipulate the binary data Oracle has an application-programming interface (API) called Oracle Call Interface (OCI) that allows a programmer to use OCI functions within C/C++ applications to manipulate the binary data. This application then becomes the interface to the Oracle database.

To store the data within the database two tables had to be created. One table called blob_header_tbl would store the header data and the other table called blob_data_tbl would store the binary data file. The blob_header_tbl table was created using the following SQL command:

```
create table blob_header_tbl (
    datetime       varchar2(12)      primary key,
    info           varchar2(80),
```

```
    sect_offset    number(5),
    byte_offset    number(5)
 );
```

The blob_data_tbl table was created using:

```
create table  blob_data_tbl (
    datetime      varchar2(12),
    bloblocator   blob,
    foreign key (date_time) references blob_header_tbl
);
```

These commands are simple enough to get started since it allows the creation of a prototype application before moving on to operations that are more complex. Reading the file and extracting the data obtain all data for the fields except bloblocator. The date time field is a concatenation of date and time values from the file. The info field contains a description of the scan. The offset fields are for extra testing. The bloblocator contains a pointer to the data file since the file would be stored in the database tablespace rather than within the rows of the database. This is a size restriction of the Oracle 8*i* database. Binary files greater than 4000 bytes cannot be stored within rows of a database table. Instead the files are stored within a tablespace of the database and a pointer to the file is stored in the rows. This pointer points to the actual binary data file within the database.

To insert data into the database tables the application design had to follow this order:

1. Connect to the database and log in.
2. Read the contents of the file.
3. Extract data from the file for insertion into tables within the database.
4. Insert the data into the database.
5. Log off and disconnect from the database.

To insert the data into the database one has to create the necessary environment that would allow for a successful operation. This environment had to be created using OCI function calls. The OCI function calls would create the environment, declare and define needed variables and log on to the database using authoritative methods. Usually a user with restrictive access will be allowed to access is to those tables. By restrictive I mean that the user most likely won’t have delete and process privileges, as this will allow the user to interfere with the process. This username and password will also be used in the application to log on to the database. Throughout this process error checking will be done on every function call and the appropriate action taken.

As stated previously the fields values of the tables were filled with data extracted from the binary file. The application had the following structure defined:

```
typedef struct header_template {
    BYTE      info[80];
    UWORD     time[3];
    UWORD     date[3];
    UWORD     sector_offset;
    UWORD     byte_offset;
} header;
header header_data;
```

BYTE was defined as an unsigned char and UWORD an unsigned short.
To assign the values from the binary data file into the struct I used the C statement

```
fread(&header_data, sizeof(header_data), 1, fp);
```

This statement allowed the needed values to be placed into the memory locations that had been set aside when the structure was defined. By reading the file into the memory location the variables are assigned their respective values. The values of these variables can then be inserted into the database using SQL statements and OCI function calls. I mentioned previously that combining the date and time variables from the structure created the primary key datetime. This combining was done using C string functions that took the numeric values from the array and built a character string. A character string was required since the primary key field in the database was of type varchar2.

Inserting the values into the header table was done by first creating a SQL statement. This statement was:

```
char *insstmt = (char *) "insert into blob_header_tbl values (:1,:2,:3,:4)";
```

The :# are place holders for the values that are going to be inserted into the table. This method of inserting data into the database is called binding by position and uses the OCIBindByPos() function. The numbers don't necessarily mean anything but is helpful when you want to know the position of a value. The OCI function that binds the data value with the position is called with the position number and the data value. After this the OCIStmtExecute() function is called that executes the SQL statement with the actual values substituted for the placeholders. The variable `insstmt` is passed as a parameter to the functions that process SQL statements. An error check is carried out and if the insert was successful, the transaction is committed. If an error occurred, the process is aborted.

Next, we have to insert the file into the blob_data_tbl. Since the file varies in size insertion into the database is done in pieces. Pieces of the file are read into a temporary variable using the C fread() function. This variable is then inserted into the database using the OCILobWrite() function. The OCILobWrite() function takes the variable, the size of the variable and an offset position. This allows the file to be written to the database exactly as it was on the disk and until the end of the file is reached. Each piece of the file is inserted within an error checking function. This ensures a continuous write to the database. This operation is also committed if no errors occurred. If an error occurs as this point in the program, a rollback command is issued to the database. The rollback erases everything that was done while the program was running. This is done because the program works on two tables. The first table contains information that pertains to the location to the data in the second table. If the data is corrupted by the insertion, the first table will contain incorrect values. So erasing the tables ensures that we do not have inconsistencies. The program now logs out of the database and clears the environment by deleting variables and returning allocated memory back the system.

In this phase of the project, the application is started manually and the files are passed

as parameters. For the next phase of the project the data will be streamed over a network connection into the computer systems. The new version of the program will then have network capabilities and most likely run as a daemon. A daemon is a term that refers to an application that runs as part of the computer system and has client/server capabilities.

Oracle provides a program called WebDB that allows users to view the data contained within a database without using SQL commands. The Oracle WebDB was created to develop a simple expense-tracking system and a user sustainable website. WebDB is a web browser-based tool that enables users to create, deploy, and monitor Web database applications and content-driven Web sites. By combining an HTML interface with a set of browser-based HTML tools, WebDB allows casual users, webmasters and application developers, to easily create useful, collaborative Web sites and Web database applications.